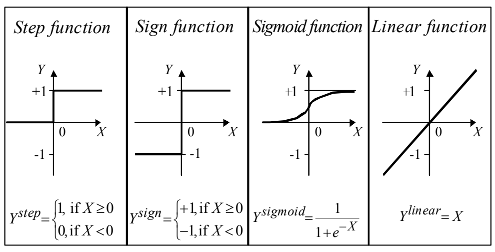
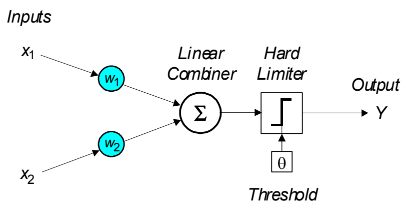
Activation functions of a neuron



$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$ $Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases}$ $Y^{sigmoid} = \frac{1}{1+e^{-X}}$ $Y^{linear} = X$



■ If at iteration p, the actual output is $Y(p)$ and the desired output is $Y_d(p)$, then the error is given by:

$$e(p) = Y_d(p) - Y(p) \quad \text{where } p = 1, 2, 3, \ldots$$

Iteration p here refers to the pth training example presented to the perceptron.

■ If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

The perceptron learning rule

$$w_i(p+1) = w_i(p) + \alpha \cdot x_i(p) \cdot e(p)$$

where $p = 1, 2, 3, \ldots$
α is the **learning rate**, a positive constant less than unity.

The perceptron learning rule was first proposed by **Rosenblatt** in 1960. Using this rule we can derive the perceptron training algorithm for classification tasks.

Learning with adaptive learning rate

To accelerate the convergence and yet avoid the danger of instability, we can apply two heuristics:

Heuristic 1 If the change of the sum of squared errors has the same algebraic sign for several consequent epochs, then the learning rate parameter, α, should be increased.

Heuristic 2 If the algebraic sign of the change of the sum of squared errors alternates for several consequent epochs, then the learning rate parameter, α, should be decreased.

■ Adapting the learning rate requires some changes in the back-propagation algorithm.

■ If the sum of squared errors at the current epoch exceeds the previous value by more than a predefined ratio (typically 1.04), the learning rate parameter is decreased (typically by multiplying by 0.7) and new weights and thresholds are calculated.

■ If the error is less than the previous one, the learning rate is increased (typically by multiplying by 1.05).

Perceptron's training algorithm

Step 1: Initialisation Set initial weights w1, w2,..., wn and threshold q to random numbers in the range [-0.5, 0.5].

If the error, e(p), is positive, we need to increase perceptron output Y(p), but if it is negative, we need to decrease Y(p).

Step 2: Activation: Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,..., $x_n(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and *step* is a step activation function.

Step 3: Weight training: Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

The weight correction is computed by the **delta rule:**

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

Step 4: Iteration: Increase iteration p by one, go back to *Step 2* and repeat the process until convergence.

Accelerated learning in multilayer neural networks

A multilayer network learns much faster when the sigmoidal activation function is represented by a hyperbolic tangent:

$$Y^{tanh} = \frac{2a}{1+e^{-bX}} - a$$

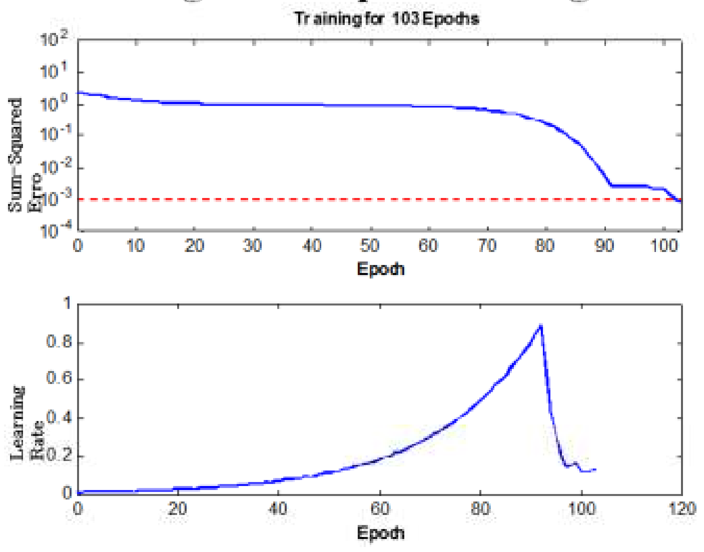
where a and b are constants. Suitable values for a and b are:
a = 1.716 and b = 0.667

We also can accelerate training by including a momentum term in the delta rule: This equation is called the generalized delta rule.

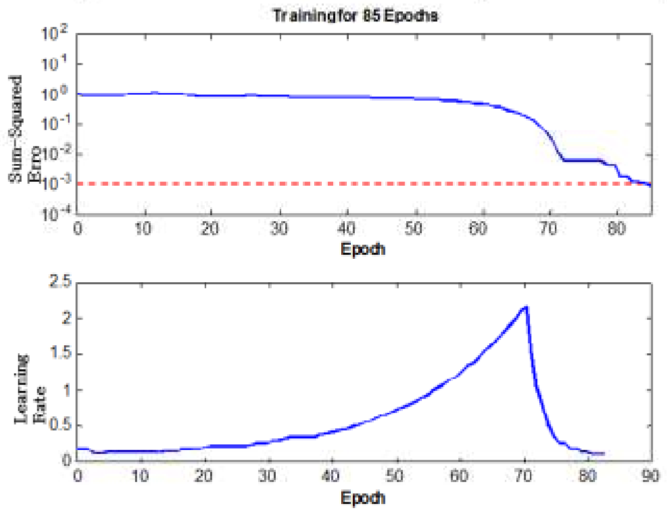
$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)$$

where b is a positive number (0 £ b < 1) called the momentum constant. Typically, the momentum constant is set to 0.95.

Learning with adaptive learning rate



Learning with momentum and adaptive learning rate



The Hopfield Network

■ Neural networks were designed on analogy with the brain. The brain's memory, however, works by association. For example, we can recognise a familiar face even in an unfamiliar environment within 100-200 ms. We can also recall a complete sensory experience, including sounds and scenes, when we hear only a few bars of music. The brain routinely associates one thing with another.

■ Multilayer neural networks trained with the back-propagation algorithm are used for pattern recognition problems. However, to emulate the human memory's associative characteristics we need a different type of network: a **recurrent neural network**.

■ A recurrent neural network has feedback loops from its outputs to its inputs. The presence of such loops has a profound impact on the learning capability of the network.

■ The stability of recurrent networks intrigued several researchers in the 1960s and 1970s. However, none was able to predict which network would be stable, and some researchers were pessimistic about finding a solution at all. The problem was solved only in 1982, when **John Hopfield** formulated the physical principle of storing information in a dynamically stable network.

■ The Hopfield network uses McCulloch and Pitts neurons with the *sign activation function* as its computing element:

$$Y^{sign} = \begin{cases} +1, \text{ if } X > 0 \\ -1, \text{ if } X < 0 \\ Y, \text{ if } X = 0 \end{cases}$$

■The current state of the Hopfield network is determined by the current outputs of all neurons, $y_1, y_2, \ldots, y_n$.

Thus, for a single-layer n-neuron network, the state can be defined by the **state vector** as:

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

■ In the Hopfield network, synaptic weights between neurons are usually represented in matrix form as follows:

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{Y}_m \mathbf{Y}_m^T - M\,\mathbf{I}$$

where M is the number of states to be memorised by the network, $\mathbf{Y}_m$ is the n-dimensional binary vector, I is n ´ n identity matrix, and superscript T denotes matrix transposition.

■ The stable state-vertex is determined by the weight matrix **W**, the current input vector **X**, and the threshold matrix **q**. If the input vector is partially incorrect or incomplete, the initial state will converge into the stable state-vertex after a few iterations.

■ Suppose, for instance, that our network is required to memorise two opposite states, (1, 1, 1) and (-1, -1, -1).

Tl

$$\mathbf{Y}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{Y}_2 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \qquad \mathbf{Y}_1^T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad \mathbf{Y}_2^T = \begin{bmatrix} -1 & -1 & -1 \end{bmatrix}$$

■ The 3 ´ 3 identity matrix **I** is

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

■ Thus, we can now determine the weight matrix as follows:

$$\mathbf{W} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} - 2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}$$

■ Next, the network is tested by the sequence of input vectors, $\mathbf{X}_1$ and $\mathbf{X}_2$, which are equal to the output (or target) vectors $\mathbf{Y}_1$ and $\mathbf{Y}_2$, respectively.

■First, we activate the Hopfield network by applying the input vector **X**. Then, we calculate the actual output vector **Y**, and finally, we compare the result with the initial input vector **X**.

$$\mathbf{Y}_1 = sign\left\{ \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\mathbf{Y}_2 = sign\left\{ \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

■ The remaining six states are all unstable. However, stable states (also called **fundamental memories**) are capable of attracting states that are close to them.

■ The fundamental memory (1, 1, 1) attracts unstable states (-1, 1, 1), (1, -1, 1) and (1, 1, -1). Each of these unstable states represents a single error, compared to the fundamental memory (1, 1, 1).

■ The fundamental memory (-1, -1, -1) attracts unstable states (-1, -1, 1), (-1, 1, -1) and (1, -1, -1).

■ Thus, the Hopfield network can act as an **error correction network**.

Storage capacity of the Hopfield network

■ **Storage capacity is** or the largest number of fundamental memories that can be stored and retrieved correctly.

■ The maximum number of fundamental memories M_{max} that can be stored in the n-neuron recurrent network is limited by

$$M_{max} = 0.15n$$

Bidirectional associative memory (BAM)

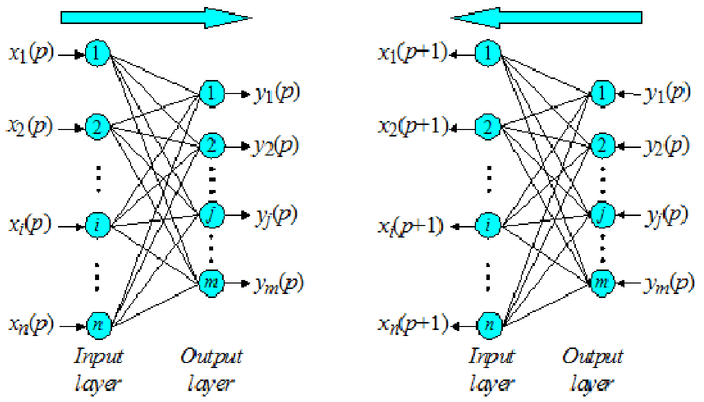
■ The Hopfield network represents an **autoassociative** type of memory - it can retrieve a corrupted or incomplete memory but cannot associate this memory with another different memory.

■ Human memory is essentially **associative**. One thing may remind us of another, and that of another, and so on. We use a chain of mental associations to recover a lost memory. If we forget where we left an umbrella, we try to recall where we last had it, what we were doing, and who we were talking to. We attempt to establish a chain of associations, and thereby to restore a lost memory.

■ To associate one memory with another, we need a recurrent neural network capable of accepting an input pattern on one set of neurons and producing a related, but different, output pattern on another set of neurons.

■ **Bidirectional associative memory (BAM)**, first proposed by **Bart Kosko**, is a heteroassociative network. It associates patterns from one set, set *A*, to patterns from another set, set *B*, and vice versa. Like a Hopfield network, the BAM can generalise and also produce correct outputs despite corrupted or incomplete inputs.

BAM operation



(*a*) Forward direction. (*b*) Backward direction.

The basic idea behind the BAM is to store pattern pairs so that when *n*-dimensional vector **X** from set *A* is presented as input, the BAM recalls *m*-dimensional vector **Y** from set *B*, but when **Y** is presented as input, the BAM recalls **X**.

To develop the BAM, we need to create a correlation matrix for each pattern pair we want to store. The correlation matrix is the matrix product of the input vector **X**, and the transpose of the output vector $\mathbf{Y}^T$. The BAM weight matrix is the sum of all correlation matrices, that is,

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{X}_m \mathbf{Y}_m^T$$

where *M* is the number of pattern pairs to be stored in the BAM.

Stability and storage capacity of the BAM

■ The BAM is **unconditionally stable**. This means that any set of associations can be learned without risk of instability.

■ The maximum number of associations to be stored in the BAM should not exceed the number of neurons in the smaller layer.

■ The more serious problem with the BAM is **incorrect convergence**. The BAM may not always produce the closest association. In fact, a stable association may be only slightly related to the initial input vector.

Unsupervised learning.

■In contrast to supervised learning, unsupervised or **self-organised learning** does not require an external teacher. During the training session, the neural network receives a number of different input patterns, discovers significant features in these patterns and learns how to classify input data into appropriate categories. Unsupervised learning tends to follow the neuro-biological organisation of the brain.

■Unsupervised learning algorithms aim to learn rapidly and can be used in real-time.

Hebbian learning

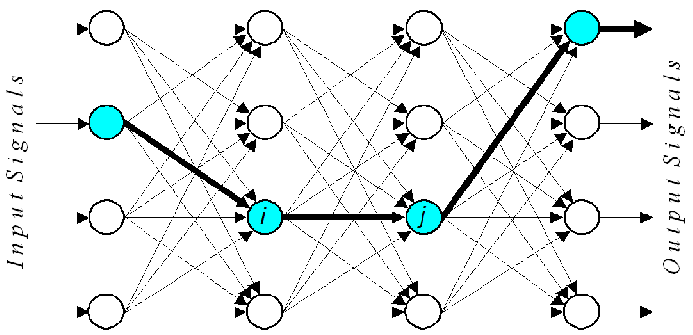
In 1949, Donald Hebb proposed one of the key ideas in biological learning, commonly known as **Hebb's Law**. Hebb's Law states that if neuron *i* is near enough to excite neuron *j* and repeatedly participates in its activation, the synaptic connection between these two neurons is strengthened and neuron *j* becomes more sensitive to stimuli from neuron *i*.

Hebb's Law can be represented in the form of two rules:

1.If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.

2.If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.

Hebb's Law provides the basis for learning without a teacher. Learning here is a **local phenomenon** occurring without feedback from the environment.



■Using Hebb's Law we can express the adjustment applied to the weight w_{ij} at iteration *p* in the following form:

$$\Delta w_{ij}(p) = F[\, y_j(p),\; x_i(p)\,]$$

As a special case, we can represent Hebb's Law as follows:

$$\Delta w_{ij}(p) = a\; y_j(p)\; x_i(p)$$

where *a* is the *learning rate* parameter. This equation is referred to as the **activity product rule**.

■Hebbian learning implies that weights can only increase. To resolve this problem, we might impose a limit on the growth of synaptic weights. It can be done by introducing a non-linear **forgetting factor** into Hebb's Law:

$$\Delta w_{ij}(p) = \alpha\; y_j(p)\; x_i(p) - \varphi\; y_j(p)\; w_{ij}(p)$$

where j is the forgetting factor.

Forgetting factor usually falls in the interval between 0 and 1, typically between 0.01 and 0.1, to allow only a little "forgetting" while limiting the weight growth.

Hebbian learning algorithm

Step 1: Initialisation. Set initial synaptic weights and thresholds to small random values, say in an interval [0, 1].

Step 2: Activation. Compute the neuron output at iteration *p*

$$y_j(p) = \sum_{i=1}^{n} x_i(p)\, w_{ij}(p) - \theta_j$$

where *n* is the number of neuron inputs, and q_j is the threshold value of neuron *j*.

Step 3: Learning. Update the weights in the network:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

where $Dw_{ij}(p)$ is the weight correction at iteration *p*.

The weight correction is determined by the generalized activity product rule:

$$\Delta w_{ij}(p) = \varphi\, y_j(p)[\lambda\; x_i(p) - w_{ij}(p)]$$

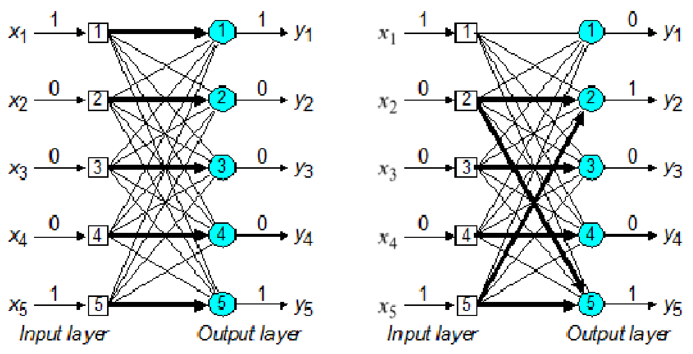
Step 4: Iteration. Increase iteration *p* by one, go back to Step 2.

Hebbian learning example

To illustrate Hebbian learning, consider a fully connected feedforward network with a single layer of five computation neurons. Each neuron is represented by a McCulloch and Pitts model with the sign activation function. The network is trained on the following set of input vectors:

$$X_1 = \begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix} \quad X_2 = \begin{bmatrix}0\\1\\0\\0\\1\end{bmatrix} \quad X_3 = \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \quad X_4 = \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix} \quad X_5 = \begin{bmatrix}0\\1\\0\\0\\1\end{bmatrix}$$

Initial and final states of the network



Initial and final weight matrices

$$\begin{array}{c|ccccc} & \multicolumn{5}{c}{\text{Output layer}} \\ \text{Input layer} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 \end{array} \qquad \begin{array}{c|ccccc} & \multicolumn{5}{c}{\text{Output layer}} \\ \text{Input layer} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 2.0204 & 0 & 0 & 2.0204 \\ 3 & 0 & 0 & 1.0200 & 0 & 0 \\ 4 & 0 & 0 & 00 & .9996 & 0 \\ 5 & 0 & 2.0204 & 0 & 0 & 2.0204 \end{array}$$

■ A test input vector, or probe, is defined as X =Transpose([1 0 0 0 1])

■ When this probe is presented to the network, we obtain:

$$Y = sign\left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \\ 0 & 0 & 1.0200 & 0 & 0 \\ 0 & 0 & 0 & 0.9996 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\1\end{bmatrix} - \begin{bmatrix}0.4940\\0.2661\\0.0907\\0.9478\\0.0737\end{bmatrix} \right\} = \begin{bmatrix}0\\1\\0\\0\\1\end{bmatrix}$$

Competitive learning

■ In competitive learning, neurons compete among themselves to be activated.

■ While in Hebbian learning, several output neurons can be activated simultaneously, in competitive learning, only a single output neuron is active at any time.

■ The output neuron that wins the "competition" is called the ***winner-takes-all*** neuron.

■ The basic idea of competitive learning was introduced in the early 1970s.

■ In the late 1980s, Teuvo Kohonen introduced a special class of artificial neural networks called **self-organising feature maps**. These maps are based on competitive learning.

What is a self-organising feature map?

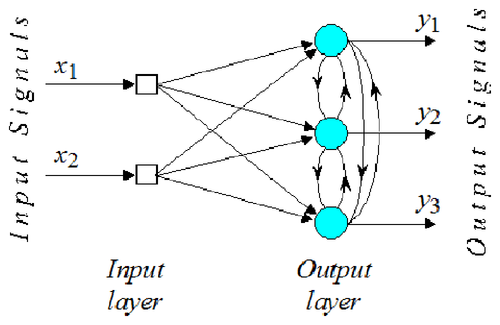
Our brain is dominated by the cerebral cortex, a very complex structure of billions of neurons and hundreds of billions of synapses. The cortex includes areas that are responsible for different human activities (motor, visual, auditory, somatosensory, etc.), and associated with different sensory inputs. We can say that each sensory input is mapped into a corresponding area of the cerebral cortex. **The cortex is a self-organising computational map in the human brain.**

The Kohonen network

■ The Kohonen model provides a topological mapping. It places a fixed number of input patterns from the input layer into a higher- dimensional output or Kohonen layer.

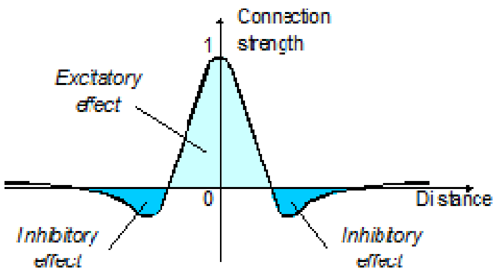
■ Training in the Kohonen network begins with the winner's neighbourhood of a fairly large size. Then, as training proceeds, the neighbourhood size gradually decreases.

■**Architecture of the Kohonen Network**



■ The lateral connections are used to create a competition between neurons. The neuron with the largest activation level among all neurons in the output layer becomes the winner. This neuron is the only neuron that produces an output signal. The activity of all other neurons is suppressed in the competition.

■ The lateral feedback connections produce excitatory or inhibitory effects, depending on the distance from the winning neuron. This is achieved by the use of a **Mexican hat function** which describes synaptic weights between neurons in the Kohonen layer.



■ In the Kohonen network, a neuron learns by shifting its weights from inactive connections to active ones. Only the winning neuron and its neighbourhood are allowed to learn. If a neuron does not respond to a given input pattern, then learning cannot occur in that particular neuron.

■ The **competitive learning rule** defines the change Dw_{ij} applied to synaptic weight w_{ij} as

$$\Delta w_{ij} = \begin{cases} \alpha(x_i - w_{ij}), & \text{if neuron } j \text{ wins the competition} \\ 0, & \text{if neuron } j \text{ loses the competition} \end{cases}$$

where x_i is the input signal and *a* is the *learning rate* parameter.

■ The overall effect of the competitive learning rule resides in moving the synaptic weight vector $\mathbf{W}_j$ of the winning neuron *j* towards the input pattern **X**. The matching criterion is equivalent to the minimum **Euclidean distance** between vectors.

■ The Euclidean distance between a pair of *n*-by-1 vectors **X** and $\mathbf{W}_j$ is defined by

$$d = \|\mathbf{X} - \mathbf{W}_j\| = \left[\sum_{i=1}^{n}(x_i - w_{ij})^2\right]^{1/2}$$

where x_i and w_{ij} are the *i*th elements of the vectors **X** and $\mathbf{W}_j$, respectively.

■ To identify the winning neuron, $j_{\mathbf{X}}$, that best matches the input vector **X**, we may apply the following condition:

$$j_{\mathbf{X}} = \min_j \|\mathbf{X} - \mathbf{W}_j\|, \qquad j = 1, 2, \ldots, m$$

where *m* is the number of neurons in the Kohonen layer.

■ Suppose, for instance, that the 2-dimensional input vector **X** is presented to the three-neuron Kohonen network,

■ The initial weight vectors, $\mathbf{W}_j$, are given by

$$\mathbf{X} = \begin{bmatrix} 0.52 \\ 0.12 \end{bmatrix} \quad \mathbf{W}_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix} \quad \mathbf{W}_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix} \quad \mathbf{W}_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

■ We find the winning (best-matching) neuron $j_{\mathbf{X}}$ using the minimum-distance Euclidean criterion:

$$d_1 = \sqrt{(x_1 - w_{11})^2 + (x_2 - w_{21})^2} = \sqrt{(0.52 - 0.27)^2 + (0.12 - 0.81)^2} = 0.73$$

$$d_2 = \sqrt{(x_1 - w_{12})^2 + (x_2 - w_{22})^2} = \sqrt{(0.52 - 0.42)^2 + (0.12 - 0.70)^2} = 0.59$$

$$d_3 = \sqrt{(x_1 - w_{13})^2 + (x_2 - w_{23})^2} = \sqrt{(0.52 - 0.43)^2 + (0.12 - 0.21)^2} = 0.13$$

■ Neuron 3 is the winner and its weight vector **W3** is updated according to the competitive learning rule.

$$\Delta w_{13} = \alpha\,(x_1 - w_{13}) = 0.1\,(0.52 - 0.43) = 0.01$$

$$\Delta w_{23} - \alpha\,(x_2 - w_{23}) - 0.1\,(0.12 - 0.21) - -0.01$$

■ The updated weight vector $\mathbf{W}_3$ at iteration (*p* + 1) is determined as:

$$\Delta w_{13} = \alpha\,(x_1 - w_{13}) = 0.1(0.52 - 0.43) = 0.01$$

$$\Delta w_{23} = \alpha\,(x_2 - w_{23}) = 0.1(0.12 - 0.21) = -0.01$$

■ The weight vector $\mathbf{W}_3$ of the wining neuron 3 becomes closer to the input vector **X** with each iteration.

Competitive Learning Algorithm

Step 1: ***Initialisation***: Set initial synaptic weights to small random values, say in an interval [0, 1], and assign a small positive value to the learning rate parameter a.

Step 2: ***Activation and Similarity Matching.*** Activate the Kohonen network by applying the input vector **X**, and find the winner-takes-all (best matching) neuron j_X at iteration p, using the minimum-distance Euclidean criterion

$$j_X(p) = \min_j \|\mathbf{X} - \mathbf{W}_j(p)\| = \left\{ \sum_{i=1}^{n} [x_i - w_{ij}(p)]^2 \right\}^{1/2},$$

$$j = 1, 2, \ldots, m$$

where n is the number of neurons in the input layer, and m is the number of neurons in the Kohonen layer.

Step 3: ***Learning.***
Update the synaptic weights

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

where Dw$_{ij}$(p) is the weight correction at iteration p. The weight correction is determined by the competitive learning rule:

$$\Delta w_{ij}(p) = \begin{cases} \alpha\left[x_i - w_{ij}(p)\right], & j \in \Lambda_j(p) \\ 0, & j \notin \Lambda_j(p) \end{cases}$$

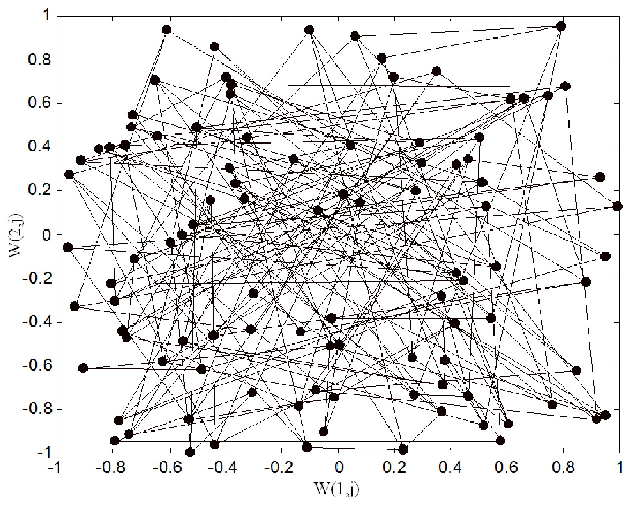
where a is the *learning rate* parameter, and L$_j$(p) is the neighbourhood function centred around the winner-takes-all neuron j_X at iteration p.

Step 4: ***Iteration.***
Increase iteration p by one, go back to Step 2 and continue until the minimum-distance Euclidean criterion is satisfied, or no noticeable changes occur in the feature map.

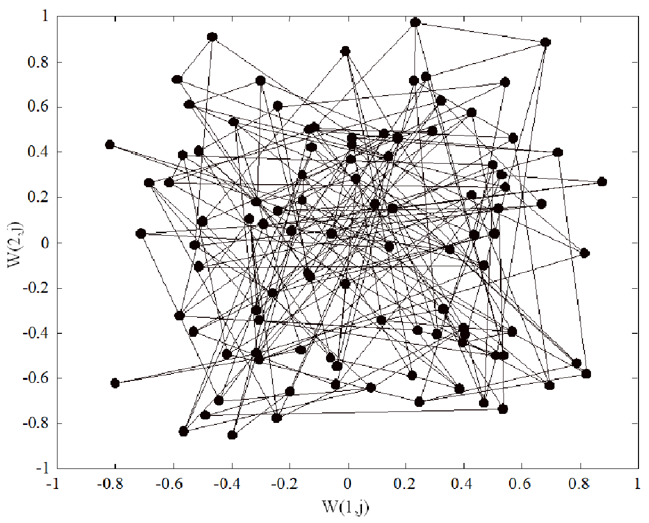
Competitive learning in the Kohonen network

- To illustrate competitive learning, consider the Kohonen network with 100 neurons arranged in the form of a two-dimensional lattice with 10 rows and 10 columns. The network is required to classify two-dimensional input vectors - each neuron in the network should respond only to the input vectors occurring in its region.
- The network is trained with 1000 two-dimensional input vectors generated randomly in a square region in the interval between –1 and +1. The learning rate parameter a is equal to 0.1.

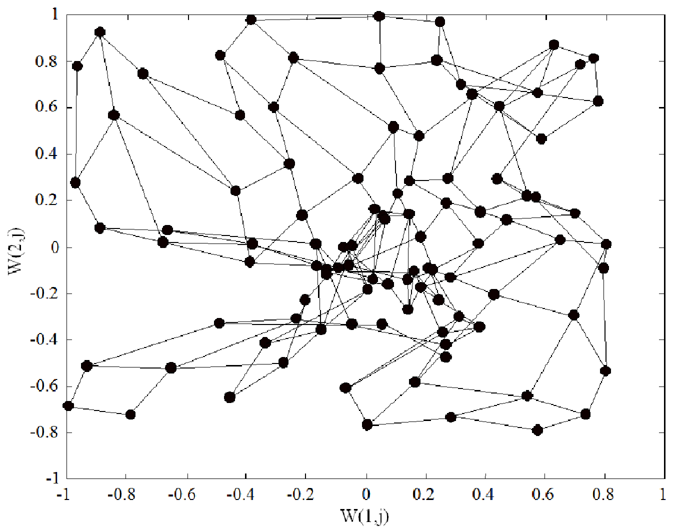
Initial random weights



Network after 100 iterations



Network after 1000 iterations



Network after 10,000 iterations